# O-Calc Pro Customization Development API

## Introduction

This document describes the development process to create a customized O-Calc Pro Plugins and Reports. O-Calc Pro Plugins can be created to perform a wide verity of functions within the O-Calc Pro interface from importing data from some type of field data collection device to exporting the data to different formats or into specialized databases. Plugins can also be developed that manipulate the data right within the O-Calc Pro interface.

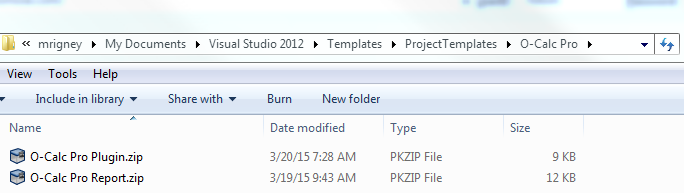
## Before You Get Started

Before you can start to develop an O-Calc Pro Plugin, a few prerequisites are necessary.

1. You must have O-Calc Pro Version 5.01 or later installed on the development machine at the following location: C:\Program Files (x86)\Osmose\O-Calc Pro\Bin
2. You must have Microsoft Visual Studio install on the development machine, this can be Visual Studio Express
3. Optionally you can install the O-Calc Pro Plugin and Report visual studio template to the following location:

C:\Users\<username>\Documents\Visual Studio 2012\Templates\ProjectTemplates

A zip file of the O-Calc Pro Visual Studio Templates can be downloaded from the O-Calc Pro User’s Website (<http://www.osmoseutilities.com/o-calcpro>). The ‘O-Calc Pro Visual Studio Templates.zip’ file contains two additional zip files in an ‘O-Calc Pro’ subfolder that would be exacted and placed right in the ProjectTemplates folder list above without further unzipping of the files.



***Note****: You must use Visual Studio 2012 or Visual Studio Express 2012 for these templates to work. This document uses examples from Visual Studio Express 2012 for Windows Desktop which can be download at* [*http://www.microsoft.com/en-us/download/details.aspx?id=34673*](http://www.microsoft.com/en-us/download/details.aspx?id=34673)

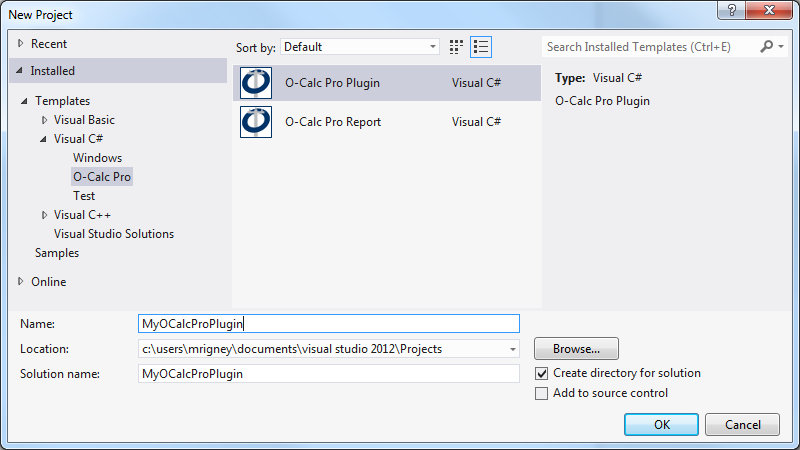
## Templates

There are two Microsoft Visual Studio templates that can help jumpstart your O-Calc Pro custom development. Within in Visual Studio Express 2012, a new development project can be started by selecting File -> New Project. The O-Calc Pro templates are Visual C# Templates with the names:

***O-Calc Pro Plugin***

***O-Calc Pro Report***

In the screenshot below, the O-Calc Pro Plugin template has been selected and a unique project name of ‘MyOCalcProPlugin’ has been entered.



The ***O-Calc Pro Plugin*** visual studio project includes three components that developer can make use of:

* **AssemblyInfo.cs** – can be used to change various general properties of the plugin
* **References** – A number of O-Calc Pro library references are include that the developer can use. These references enable the normal Visual C# IntelliSence drop-down menus for the developer.
* **Plugin.cs** – This is the main source file of the ***O-Calc Pro Plugin***. Within this file, the developer can determine the type of plugin (Menu Item, Docked Tab, or both) and write the code to perform the plugin tool’s operation.

The ***O-Calc Pro Report*** visual studio project includes four components that the developer can make use of:

* **AssemblyInfo.cs** – can be used to change various general properties of the report
* **References** – A number of O-Calc Pro library references are include that the developer can use. These references enable the normal Visual C# IntelliSence drop-down menus for the developer.
* **Helpers.cs** – This a helper source file of the ***O-Calc Pro Report*** template. This source file contains various helper methods that aid in formatting the report. Items such as creating and formatting tables, color coding text based on loading values, and manipulating images and graphics.
* **Report.cs** – This is the main source file of the ***O-Calc Pro Report*** template. This source file enables the developer to define the various aspects of the report format including colors, fonts, and data to be displayed. There are also methods to implement more than one report subtype and determining the structure type that is applicable for this report.

## O-Calc Pro Plugin Example – “Hello World”

The default ***O-Calc Pro Plugin*** template will create a message box showing the message “Hello World”. Below are some explanations of the various code components of this simple plugin example.

### Defining the Plugin Type:

/// <summary>

/// Declare the type of plugin as one of:

/// DOCKED\_TAB

/// MENU\_ITEM

/// BOTH\_DOCKED\_AND\_MENU

/// CLEARANCE\_SAG\_PROVIDER

/// </summary>

public PLUGIN\_TYPE Type

{

get

{

return PLUGIN\_TYPE.BOTH\_DOCKED\_AND\_MENU;

}

}

There are four different plug-in types that can be defined. In the code snippet above, the plugin type as been defined as both a ‘Menu Item’ and a new ‘Docked Tab’.

### Naming the Plugin:

/// <summary>

/// Declare the name of the plugin usd for synthesizing the registry keys ect

/// </summary>

public String Name

{

get

{

return "MyOCalcProPlugin";

}

}

This will be the name of the menu item as it appears in the O-Calc Pro interface. To change the name of the resultant compiled plugin file that gets built to the location C:\Program Files (x86)\Osmose\O-Calc Pro\bin\Plugins, you would need to change the name of the Visual Studio Project. Note: the name of the compiled plugin file is what will be displayed in the O-Calc Pro ‘Manage Plugins’ list.

### Specifying the Docked Tab Parameters :

/// <summary>

/// Add a tabbed form to the tabbed window (if the plugin type is

/// PLUGIN\_TYPE.DOCKED\_TAB

/// or

/// PLUGIN\_TYPE.BOTH\_DOCKED\_AND\_MENU

/// </summary>

/// <param name="pPPLMain"></param>

public void AddForm(PPL\_Lib.PPLMain pPPLMain)

{

. . .

}

PluginForm cForm = null;

class PluginForm : WeifenLuo.WinFormsUI.Docking.DockContent

{

public PluginForm()

{

this.Name = "pluginForm";

this.Text = "Plugin";

System.Windows.Forms.Label helloLabel = new System.Windows.Forms.Label();

helloLabel.AutoSize = true;

helloLabel.Name = "helloLabel";

helloLabel.Text = "Hello World";

helloLabel.Location = new System.Drawing.Point(20, 20);

helloLabel.Visible = true;

Controls.Add(helloLabel);

}

public Guid cGuid;

protected override string GetPersistString()

{

return cGuid.ToString();

}

Here the developer can specify the various parameters associated with the docked tab, such as tab name, display text, and other features of the new docked tab that will appear within the O-Calc Pro interface.

### Defining the location of the Menu Item:

public void AddToMenu(PPL\_Lib.PPLMain pPPLMain, System.Windows.Forms.ToolStrip pToolStrip)

{

//save the reference to the O-Calc Pro main

cPPLMain = pPPLMain;

//create the toolstrip button

cToolStripMenuItemButton = new ToolStripMenuItem(Name);

cToolStripMenuItemButton.AutoToolTip = true;

cToolStripMenuItemButton.ToolTipText = Description;

//find the dropdown menu we want to add the toolsrip button to

int itemindex = 0;

System.Diagnostics.Debug.Assert(pToolStrip.Items[itemindex] is ToolStripDropDownButton);

if (pToolStrip.Items[itemindex] is ToolStripDropDownButton)

{

ToolStripDropDownButton tsb = pToolStrip.Items[itemindex] as ToolStripDropDownButton;

System.Diagnostics.Debug.Assert(tsb.Text == "&File");

tsb.DropDownItems.Insert(tsb.DropDownItems.Count - 1, cToolStripMenuItemButton);

tsb.DropDownItems.Insert(tsb.DropDownItems.Count - 1, new ToolStripSeparator());

//add an event handler when the dropdown menu is opened to allow us

//to enable or disble the toolstrip button (optional)

tsb.DropDownOpened += tsb\_DropDownOpened;

}

In the default code snippet above, the new menu item is placed at the bottom of the ‘File’ drop down menu. This menu item will be disabled until a pole has been loaded within the application.

/// <summary>

/// The menu containing our tool is being displayed. Optionally

/// enable or disable the toolstrip button depending on

/// our criteria

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void tsb\_DropDownOpened(object sender, EventArgs e)

{

bool enabled = false;

if (cPPLMain != null)

{

enabled = (cPPLMain.GetMainStructure() is PPLPole);

}

cToolStripMenuItemButton.Enabled = enabled;

}

### Perform the Plugin Operation

/// <summary>

/// Perform the plugin tool's operation

/// </summary>

private void DoPluginOperation()

{

try

{

PPLMessageBox.Show("Hello World!");

}

catch (Exception ex)

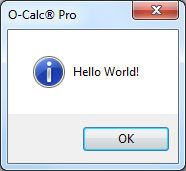
{

PPLMessageBox.Show(ex.Message, "Error in " + Name);

}

}

In this example, clicking on the plugin menu item found in the File dropdown, will display the “Hello World” message box.



## O-Calc Pro Report Example – “My First Report”

The default ***O-Calc Pro Report*** template will create a simple report with default header and a single table of pole capacity utilization values. Below are some explanations of the various code components of this simple report example.

### Defining the Report Format, Fonts, and Colors:

public static class ReportFont

{

public static double cPageWidth = 6.25;

public static string cFontName = "Arial";

public static MD.Unit cFontSizeLarge = MD.Unit.FromPoint(9.75);

public static MD.Unit cFontSizeSmall = MD.Unit.FromPoint(7.85);

}

public partial class Report : PPL\_Lib.ReportTemplate

{

//globally defined colors and fonts

MD.Color Yellow = MD.Colors.LightYellow;

MD.Color White = MD.Colors.White;

MD.ParagraphAlignment JustifyRight = MD.ParagraphAlignment.Right;

MD.ParagraphAlignment JustifyLeft = MD.ParagraphAlignment.Left;

MD.ParagraphAlignment JustifyCenter = MD.ParagraphAlignment.Center;

MD.Font SmallFont = new MD.Font(ReportFont.cFontName, ReportFont.cFontSizeSmall);

MD.Font StandardFont = new MD.Font(ReportFont.cFontName, ReportFont.cFontSizeLarge);

FontModifier Bold = new FontModifier(true, false, false);

This font, color, and paragraph parameters will be used in the main body of the report. These parameters do not define the header section of the report.

### Define the Name (or Names) of the Report(s):

//implement more than one additional report subtype (total will be this number plus the master)

public override int SubtypesCount { get { return 0; } }

public override REPORT\_TYPE Type { get { return REPORT\_TYPE.INTEGRATED; } }

public override String Name { get { return "My First Report"; } }

public override String Description { get { return Name; } }

//make sure to implement ToString to show report name

public override String ToString() { return Name; }

If you are defining more than one report time, change the SubtypesCount and create a separate name for each subtype. In this example, a single report with the name “My First Report” is being created.

### Define the structure type that is applicable for the report:

//determine if this report is applicable for a structure

public override bool IsApplicable(PPLMain pPPLMain, PPLPole pPole)

{

if (Filtered) return false;

if (pPole is PPLWoodPole)

{

return true;

}

return false;

}

public override bool IsApplicableForType(PPLMain.POLE\_TYPE pType)

{

if (Filtered) return false;

return (pType == PPLMain.POLE\_TYPE.WOOD);

}

This section of the report will enable the user to determine what structure type this report will run on. Structure types include WOOD, STEEL, COMPOSITE, CONCRETE, SEGMENTED, MULTI-POLE, and/or LATTICE. In the example code above, this report will show up in the O-Calc Pro interface as an applicable report only for pole type of WOOD.

### Add the Pole and format the Report as Desired

public override void AddPole(PPLPole pPole, String pPoleName)

{

String pplx = cPPLMain.ReportingLoadedPolePath;

try

{

pplx = System.IO.Path.GetFileName(pplx);

}

catch { }

PoleLoadInfo poleInfoMaster = cPPLMain.CurrentPoleLoadInfo;

ReportItems reportItems = new ReportItems(poleInfoMaster, cPPLMain, pplx);

ReportItems.ReportParameters RP = reportItems.cReportParameters;

CreateTable(ReportFont.cPageWidth, 4);

cCurrentTable.Format.Font = SmallFont.Clone();

cCurrentTable.Borders.Visible = false;

//Header Row

AddRow(Cell("Pole Capacity Utilization (%)", 2, MD.Colors.White, JustifyLeft, StandardFont, Bold),

Cell("Height" + "\n(" + RP.Units\_Len\_Large\_Abrev + ")", MD.Colors.White, JustifyRight, StandardFont, Bold),

Cell("Wind Angle" + "\n(deg)", MD.Colors.White, JustifyRight, StandardFont, Bold));

cCurrentTable.RightPadding = MD.Unit.FromInch(0.125);

//First Row

AddRow(Cell("Maximum", JustifyLeft),

Cell(Math.Round(RP.Sum\_MCU, 1).ToString("0.0"), Yellow, JustifyRight, StandardFont, Bold,

GetColorFromValueAndTolerance(RP.Sum\_MCU, 80, 90, 99.5)),

Cell(Math.Round(RP.Sum\_MCU\_Height, 1).ToString("0.0"), JustifyRight),

Cell(Math.Round(RP.Sum\_MCU\_Wind\_Angle, 1).ToString("0.0"), JustifyRight));

//Second Row

AddRow(Cell("Groundline", JustifyLeft),

Cell(Math.Round(RP.Sum\_GLCU, 1).ToString("0.0"), Yellow, JustifyRight, StandardFont, Bold,

GetColorFromValueAndTolerance(RP.Sum\_GLCU, 80, 90, 99.5)),

Cell(Math.Round(RP.Sum\_GLCU\_Height, 1).ToString("0.0"), JustifyRight),

Cell(Math.Round(RP.Sum\_GLCU\_Wind\_Angle, 1).ToString("0.0"), JustifyRight));

//Third Row

AddRow(Cell("Vertical", JustifyLeft),

Cell(Math.Round(RP.Sum\_Vertical\_Buckling\_CU, 1).ToString("0.0"), Yellow, JustifyRight, StandardFont, Bold,

GetColorFromValueAndTolerance(RP.Sum\_Vertical\_Buckling\_CU, 80, 90, 99.5)),

Cell(Math.Round(RP.Sum\_Vertical\_Buckling\_Column\_Height, 1).ToString("0.0"), JustifyRight),

Cell(Math.Round(RP.Sum\_Vertical\_Buckling\_Wind\_Angle, 1).ToString("0.0"), JustifyRight));

cCurrentTable.BottomPadding = MD.Unit.FromInch(0.03);

cCurrentTable.TopPadding = MD.Unit.FromInch(0.03);

cCurrentTable.SetEdge(0, 0, cCurrentTable.Columns.Count, cCurrentTable.Rows.Count, MDT.Edge.Box, MD.BorderStyle.Single, 1, MD.Colors.Black);

cCurrentTable.SetEdge(0, 0, cCurrentTable.Columns.Count, 1, MDT.Edge.Box, MD.BorderStyle.Single, 1, MD.Colors.Black);

FinalizeTable();

cDocument.LastSection.AddParagraph("\n");

}